

APPLICATION
FOR
UNITED STATES LETTERS PATENT

TITLE: METHOD AND SYSTEM FOR ACCESSING
EXTERNALLY-DEFINED OBJECTS FROM AN ARRAY-
BASED MATHEMATICAL COMPUTING ENVIRONMENT

APPLICANT: DAVID A. FOTI AND CHARLES G. NYLANDER

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL235811129US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

March 3, 2000

Date of Deposit

Signature

Vince Defante

Typed or Printed Name of Person Signing Certificate

00519287-030300

METHOD AND SYSTEM FOR ACCESSING EXTERNALLY-DEFINED OBJECTS FROM AN ARRAY-BASED MATHEMATICAL COMPUTING ENVIRONMENT

TECHNICAL FIELD

The invention relates generally to mathematical computer programs.

5

BACKGROUND

A conventional mathematical tool, such as such as MATLAB™ from MathWorks™, Inc., of Natick, Massachusetts, provides a comprehensive technical computing environment for performing numerical linear algebraic calculations, solving ordinary differential equations, analyzing data, and visualizing solutions to complex mathematical formulas by generating graphs or other images. The computing environment often provides a high-level programming language that includes a variety of operators and programming commands.

10
15
20
25
Engineers use such mathematical tools for a variety of applications such as designing complex mechanical and electrical control systems, solving optimization problems and performing statistical analysis. In addition, engineers often use mathematical tools in conjunction with a simulation tool for defining and simulating complex mathematical models. For example, manufacturers of mechanical and electronic systems, e.g., cars and integrated circuits, use simulation tools to help them design their products. These tools allow designers to build and test mathematical models of their systems before building a physical prototype. Commercial simulation models can be extremely complex and may include thousands of interconnected functional blocks. Using a simulation tool, a designer can simulate and observe changes in a model over a period of time, typically represented as a series of discrete instants, called time steps, such as 1 millisecond, 1 second, 2 hours, etc. Starting from a set of initial conditions, specified by the designer, the simulation tool drives the model and determines the state of the model at various time steps.

Most technical computing environments provided by conventional mathematical tools are “array-based” such that data types are primarily represented as two-dimensional arrays. In other words, these computing environments do not distinguish between a scalar, a vector, or a matrix. As a result, it is difficult to interface the technical computing environment to an

object-oriented environment, such as Java. Because the technical computing environment does not distinguish between scalars, vectors and matrices, it is difficult to invoke methods that have the same name and are only distinguishable by the data types of their input parameters. In addition, it is difficult to translate data from the array-based computing
5 environment of the mathematical tool to the object-oriented environment.

SUMMARY OF THE INVENTION

In general, the invention provides a method and apparatus, including a computer program apparatus, which facilitate invoking methods of objects defined within an object-oriented environment from a technical computing environment provided by a mathematical
10 tool. In particular, the invention is directed to techniques for invoking methods of objects defined in an object-oriented environment, such as a Java environment, from an array-based computing environment often used in conventional mathematical tools. When a method is invoked from the computing environment, the techniques automatically compare the input parameters, which are typically arrays, with data types accepted by methods defined within
15 the object-oriented environment. Based on this comparison, the invention automatically selects a method that best accepts the input arrays. The invention, therefore, allows a user to easily invoke methods from external objects, such as Java objects, directly from the technical computing environment of the mathematical tool.

In one aspect, the invention is directed to a technique for invoking a method defined
20 within an object-oriented environment. According to the technique, a list of method signatures corresponding to a particular class and method name is retrieved from the object-oriented environment. Each signature uniquely identifies a corresponding method and lists the method's name and any data types received by the method. After the list is retrieved, the method signatures are ranked by comparing the data types of the signatures with the data
25 types of the input parameters received from the technical computing environment of the mathematical tool. Based on the ranking, one of the method signatures is selected and the corresponding method within the object-oriented environment is invoked unless no suitable method is found, in which case an error condition is raised.

In another aspect, the invention is directed to a computer program, such as a
30 mathematical tool, having instructions suitable for causing a programmable processor to

retrieve a list of method signatures from the object-oriented environment. The computer program ranks the method signatures, selects one of the method signatures according to the ranking; and invokes the corresponding method within the object-oriented environment corresponding to the method signature.

5 In yet another aspect, the invention is directed to a computer system having an object-oriented environment and a mathematical tool executing thereon. The object-oriented environment includes an interface for identifying methods provided by objects defined within the object-oriented environment. The mathematical tool includes a calculation workspace, a command interpreter, and a signature selector. When the command interpreter encounters a
10 reference to a method implemented by an object defined within the object-oriented environment, the command interpreter instructs the signature selector to access the interface of the object-oriented environment to retrieve and rank a list of signatures corresponding to methods defined within the object-oriented environment. The command interpreter invokes one of the methods as a function of the ranking.

15 The details of various embodiments of the invention are set forth in the accompanying drawings and the description below. Other features and advantages of the invention will become apparent from the description, the drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating a system in which a mathematical tool invokes
20 a method of an object defined within an object-oriented computing environment.

Figure 2 is a flow chart illustrating one embodiment of a process, suitable for implementation in a computer program, in which the mathematical tool invokes the method of the object.

Figure 3 illustrates one embodiment of a two-dimensional table that stores data types
25 supported by an object-oriented environment ordered by preference.

Figure 4 illustrates one embodiment of a conversion table suitable for converting data types from an object-oriented environment to an array-based computing environment of a mathematical tool.

Figure 5 is a block diagram illustrating a programmable processing system suitable
30 for implementing and performing the apparatus and methods of the invention.

DETAILED DESCRIPTION

Figure 1 is a block diagram illustrating a system 100 in which mathematical tool 102 invokes an object 110 in an object-oriented environment such as Java environment 120. Mathematical tool 102 provides a technical computing environment 108 for performing a wide variety of numerical calculations and data analysis operations. Computing environment 108 of mathematical tool 102 is "array-based" such that most data types are represented as arrays of at least two dimensions.

Computing environment 108 of mathematical tool 102 is an interpreted environment that supports a high-level programming language having a variety of operators and programming commands. As the user enters instructions, command interpreter 104 interactively interprets and executes each instruction. Calculation workspace 106 provides a storage area for variables, input data, and resultant data. The user can, for example, define a square matrix within calculation workspace 106 using a single command. The user can directly manipulate the matrix, using one command to find its inverse, another command to find its transpose, or another command to learn its determinant.

In an object-oriented environment, such as Java environment 120, objects 110 are modules of computer code that specify the data types of a data structure, and also the kinds of operations (or "methods") that can be applied to the data structure. Each object 110 has a corresponding "class" that may be thought of as a prototype that defines the data structures and methods common to all objects of a certain kind. Objects 110 are created at run-time in accordance with their class definition. Thus, each object 110 is a unique instance, referred to as an instantiation, of its corresponding class.

Within a class, each method having the same name must have a different number of inputs, or one or more inputs must differ in data type. Each method has a "signature", which is a unique representation of the method's name and the number and type of each input and output parameter of the method. The method signature is used to distinguish between methods having the same name. For example, in a Java signature, the data types *boolean*, *byte*, *char*, *short*, *int*, *long*, *float*, and *double*, are represented in the signature by a single letter: Z, B, C, S, I, J, F, and D, respectively. For all other data types, the signature is an expression of the form "Lclass-name;" where class-name is the name of the corresponding

Java class but with dots replaced by the slash character. A *void* return data type is indicated as a V. Thus, the signature for the method:

```
void sampleMethod(int arg1, double arg2, java.lang.String arg3)
```

has a corresponding signature:

```
(IDLjava/lang/String;)V
```

At the core of Java environment 120 is virtual machine 122, which provides a self-contained operating environment that is machine independent. Java objects 110 execute within virtual machine 122 regardless of the underlying operating system or hardware and represent any class that virtual machine 122 can see within its scope of execution.

The invention allows a user to easily invoke methods of objects 110 from mathematical tool 102. This allows the user to exploit the rich functionality offered by Java environment 120. For example, the user can invoke Java objects 110 in order to quickly design a graphical user interface (GUI). In addition, the user can use Java objects 110, such as timers and events, within calculation workspace 106. For example, the user can define and access Java objects 110 from within calculation workspace 106 as follows:

```
jstr = java.lang.String('Hello World');  
imFilter.setPixels(5,5,100,100,cm,X,0,100);
```

In order for mathematical tool 102 to provide a way for users to invoke objects 110 and their corresponding methods, command interpreter 104 invokes signature selector 112 that automatically determines the appropriate signature of the requested method for invocation. When the user invokes a method provided by one of the objects 110, command interpreter 105 passes signature selector 112 a name of the method and any input parameters to pass to the method. Because the input parameters are defined in native data types supported by technical computing environment 108, the parameters are often in the form of an array having any number of dimensions. As described in detail below, signature selector 112 automatically selects a method from object-oriented environment 120 that is best able to receive the data from the array inputs.

More specifically, signature selector 112 uses a set of classes within Java environment 120, referred to herein as selection support classes 124, to interrogate Java environment 120. Signature selector 112 passes selection support classes 124 a method name and the name of its corresponding class. Based on the class name and method name,
5 selection support classes 124 determine a set of matching method signatures available within object-oriented environment 120. In order to communicate with selection support classes 124, signature selector 112 uses Java native interface (JNI) 126, which is a programming interface, or API, that allows programs written in C or C++ to invoke Java methods based on a method signature. Signature selector 112 determines and returns the signature of the
10 method available within object-oriented environment 120 that is best able to receive the data from the array inputs. If no suitable methods are found, signature selector 112 returns a null signature. Command interpreter 104 uses selected signature 112 to directly invoke the corresponding object 110 and execute the desired method.

Figure 2 is a flow chart illustrating one embodiment of a process 200, suitable for
15 implementation in a computer program application, in which mathematical tool 102 (FIG. 1) invokes a method of a Java object 110 defined within Java environment 120. When the user seeks to invoke a method provided by one of the Java objects 110, command interpreter 104 invokes signature selector 112 to automatically determine the appropriate signature of the requested method. Selection support classes 124 interrogate java environment 120 and
20 compiles a list of method signatures having names similar to the requested method and having a matching class name (step 203).

Next, signature selector 112 calculates a “fitness ranking” for each method signature of the list (step 205). The fitness ranking indicates how well the input data types of each method match the input parameters passed from calculation workspace 106, i.e., how well
25 the method is able to receive the data from the input arrays. In order to calculate a signature’s fitness ranking, signature selector 112 generates a “preference value” for each data type specified by the signature by comparing each data type with the input parameters received from workspace 106 (step 207). For each data type specified by the signature, signature selector 112 references preference table 116, which maps data types of computing
30 environment 108 to acceptable data types of Java environment 120 ordered by preference.

Figure 3 illustrates one embodiment of a two-dimensional preference table 116. Each row of selection preference table 116 corresponds to a unique array type supported by computing environment 108. For example, row 310 corresponds to input parameters of type *array of doubles* and lists preferred data types for Java environment 120 as *double, float, long, integer, byte* and *boolean* ordered from best fit to worst fit. Thus, for input parameters of type *array of doubles*, signature selector 112 generates a preference value by determining the location of the corresponding signature data type within row 310. If the corresponding data type defined by the signature is not found within row 310 then signature selector 112 rejects the signature from the list.

In calculating the preference value for an input data type defined by the signature, signature selector 112 also considers whether the data type of the signature and the corresponding input parameter received from calculation workspace 106 are both classes. If so, signature selector 112 updates the preference value for that signature data type as a function of how many levels separate the two classes within a class hierarchy (step 209).

Next, signature selector 112 compares the number of dimensions of the input array received from calculation workspace 106 against the number of dimensions of the Java input data type defined by the current signature (step 211). If the number of dimensions of the input array is larger than the number of dimensions of the Java data type, the signature is rejected because the input array cannot fit into any Java parameter that can be passed to the Java method. If the number of dimensions of the Java data type is larger than that of the input array, the input array is promoted by adding dimensions of length 1. However, because the match is not perfect, the corresponding preference value is adjusted in proportion to the degree of difference between the number of dimensions of the signature data type and the number of dimensions of the input array. In one implementation, signature selector 112 does not count dimensions of length 1 when determining the number of dimensions. For example, a 5x1 array is considered to have a single dimension.

After calculating a preference value for each data type specified by the signature, signature selector 112 calculates the fitness ranking for the signature according to the individual preference values for the data types defined by the signature (step 213). It should be noted, however, that signature selector 112 need not explicitly store the calculated preference value for each parameter of the signature. To the contrary, signature selector 112

can calculate the fitness ranking for the signature while iterating over the data types defined by the signature. In one implementation, signature selector 112 initializes a fitness ranking, *Fitness_Ranking*, to a large number, such as 20, and updates the ranking for each parameter of the current method signature. For example, consider the following method invoked from within workspace 106:

```
f = javaObject.example_method(parameter1, parameter2);
```

Assume parameter1 of the method is a 1x1 *array of doubles* and parameter2 is a 15x1 *array of characters*. Consider a method signature defining a first data type *long* and a second data type *array of char* having two dimensions. Signature selector 112 subtracts two from *Fitness_Ranking* because, in selection preference table 116, the data type *long* is third of the data types preferred for an input data type *array of doubles*. Next, signature selector 112 determines that the data type *array of char* is in the most preferred data type for an input data type of *array of characters* and, therefore, does not adjust *Fitness_Ranking*.

Because the parameters are not objects, signature selector 112 does not adjust *Fitness_Ranking* based on differences in class level. Next, signature selector 112 considers the dimensions and determines that the first data type of the signature, *long*, is a perfect match dimensionally for a 1x1 *array of doubles*. Thus, signature selector 112 does not update *Fitness_Ranking*. However, the two dimensional *array of char* is one dimension greater than the 15x1 *array of characters*, so signature selector 112 adjusts *Fitness_Ranking* by one, resulting in a final value for *Fitness_Ranking* of 17.

After calculating fitness rankings for each potential signature, signature selector 112 selects the signature having the highest ranking, unless all of the signatures have been rejected as being unsuitable (step 215). Signature selector 112 returns the selected signature to command interpreter 104.

Upon receiving a valid signature, command interpreter 104 invokes the corresponding Java method within object-oriented environment 120 (step 217). Invoking the Java method has two parts: (1) converting input array parameters from computing environment 108 to input parameters defined by the signature, and (2) converting parameters returned by the method into suitable data types defined within computing environment 108.

In converting an input array to a data type defined by the signature, argument converter 114 of signature selector 112 generates a Java variable according to the signature and copies data from the input array to newly created variable. Signature selector 112 returns the newly created variable to command interpreter 104 for use when invoking the
5 corresponding method.

If the invoked method has a return value, signature selector 112 examines the signature and determines the dimensions of the return value. Argument converter 114 of signature selector 112 then references conversion table 118 and creates a return variable within workspace 106 for holding the return data. Figure 4 illustrates one embodiment of a
10 conversion table 118 suitable for converting data types from an object-oriented environment, such as Java environment 120, to array-based computing environment 108 of mathematical tool 102. If the return parameter is scalar, then the return variable primarily defaults to a 1x1 array of type *double precision floating point*. If the Java return value is a rectangular multi-dimensional array, signature selector 112 creates an array having the same number of
15 dimensions as the return value and having the same data type. If, however, the return value is an *array of arrays* in which the inner arrays have different lengths, then signature selector 112 creates an *array of arrays* because it cannot create a single, rectangular array. Similarly, signature selector 112 applies this technique for return values of having greater dimensions. After creating the return variable in workspace 106, signature selector 112 copies data from
20 the return parameters directly into the return variable and passes the return variable to command interpreter 104.

Various embodiments have been described of a method and system that facilitates invoking methods of objects defined within an object-oriented environment from an array-based technical computing environment often used in conventional mathematical tools. The
25 invention can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. Apparatus of the invention can be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor; and method steps of the invention can be performed by a programmable processor executing a program of instructions to
30 perform functions of the invention by operating on input data and generating output. The invention can be implemented advantageously in one or more computer programs that are

executable within an operating environment of a programmable system including at least one programmable processor (computer) coupled to receive data and instructions from, and to transmit data and instructions to, a data storage system, at least one input device, and at least one output device.

5 An example of one such type of computer is shown in Figure 5, which shows a block diagram of a programmable processing system (system) 500 suitable for implementing or performing the apparatus or methods of the invention. As shown in Figure 5, the system 500 includes a processor 512 that in one embodiment belongs to the PENTIUM® family of microprocessors manufactured by the Intel Corporation of Santa Clara, California. However,
10 it should be understood that the invention can be implemented on computers based upon other microprocessors, such as the MIPS® family of microprocessors from the Silicon Graphics Corporation, the POWERPC® family of microprocessors from both the Motorola Corporation and the IBM Corporation, the PRECISION ARCHITECTURE® family of microprocessors from the Hewlett-Packard Company, the SPARC® family of
15 microprocessors from the Sun Microsystems Corporation, or the ALPHA® family of microprocessors from the Compaq Computer Corporation. System 500 represents any server, personal computer, laptop or even a battery-powered, pocket-sized, mobile computer known as a hand-held PC or personal digital assistant (PDA).

System 500 includes system memory 513 (including read only memory (ROM) 514
20 and random access memory (RAM) 515, which is connected to the processor 512 by a system data/address bus 516. ROM 514 represents any device that is primarily read-only including electrically erasable programmable read-only memory (EEPROM), flash memory, etc. RAM 515 represents any random access memory such as Synchronous Dynamic Random Access Memory.

25 Within the system 500, input/output bus 518 is connected to the data/address bus 516 via bus controller 519. In one embodiment, input/output bus 518 is implemented as a standard Peripheral Component Interconnect (PCI) bus. The bus controller 519 examines all signals from the processor 512 to route the signals to the appropriate bus. Signals between the processor 512 and the system memory 513 are merely passed through the bus controller
30 519. However, signals from the processor 512 intended for devices other than system memory 513 are routed onto the input/output bus 518.

Various devices are connected to the input/output bus 518 including hard disk drive 520, floppy drive 521 that is used to read floppy disk 551, and optical drive 522, such as a CD-ROM drive that is used to read an optical disk 552. The video display 524 or other kind of display device is connected to the input/output bus 518 via a video adapter 525. Users enter commands and information into the system 500 by using a keyboard 540 and/or pointing device, such as a mouse 542, which are connected to bus 518 via input/output ports 528. Other types of pointing devices (not shown in Figure 5) include track pads, track balls, joysticks, data gloves, head trackers, and other devices suitable for positioning a cursor on the video display 524.

As shown in Figure 5, the system 500 also includes a modem 529. Although illustrated in Figure 5 as external to the system 500, those of ordinary skill in the art will quickly recognize that the modem 529 may also be internal to the system 500. The modem 529 is typically used to communicate over wide area networks (not shown), such as the global Internet. Modem 529 may be connected to a network using either a wired or wireless connection. System 500 is coupled to remote computer 549 via local area network 550.

Software applications 536 and data are typically stored via one of the memory storage devices, which may include the hard disk 520, floppy disk 551, CD-ROM 552 and are copied to RAM 515 for execution. In one embodiment, however, software applications 536 are stored in ROM 514 and are copied to RAM 515 for execution or are executed directly from ROM 514.

In general, the operating system 535 executes software applications 536 and carries out instructions issued by the user. For example, when the user wants to load a software application 536, the operating system 535 interprets the instruction and causes the processor 512 to load software application 536 into RAM 515 from either the hard disk 520 or the optical disk 552. Once one of the software applications 536 is loaded into the RAM 515, it can be used by the processor 512. In case of large software applications 536, processor 512 loads various portions of program modules into RAM 515 as needed.

The Basic Input/Output System (BIOS) 517 for the system 500 is stored in ROM 514 and is loaded into RAM 515 upon booting. Those skilled in the art will recognize that the BIOS 517 is a set of basic executable routines that have conventionally helped to transfer information between the computing resources within the system 500. Operating system 535

5 USER.DAT and SYSTEM.DAT, located on a permanent storage device such as an internal disk.

10 application is intended to cover any adaptation or variation of the present invention. It is intended that this invention be limited only by the claims and equivalents thereof.

[illegible]